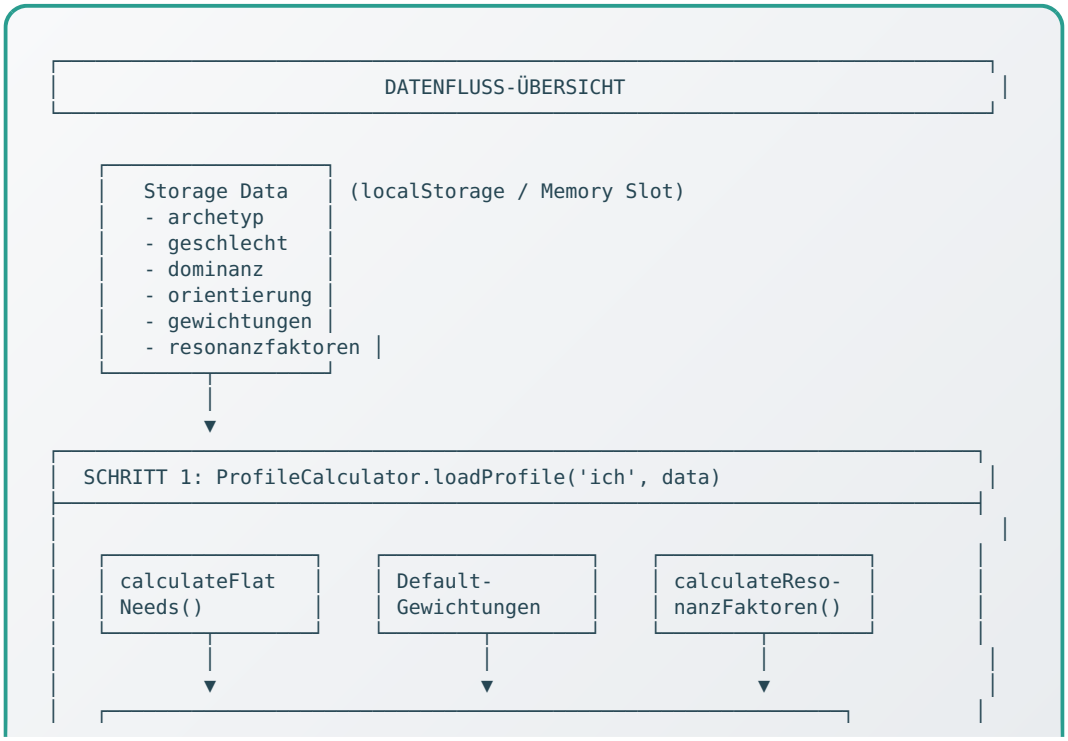
ProfileCalculator Datenfluss-Dokumentation

Technische Dokumentation des Profil-Lade- und Berechnungsprozesses

Übersicht

Der **ProfileCalculator** ist das zentrale Modul für die Berechnung und Verwaltung von Profildaten. Diese Dokumentation beschreibt den vollständigen Datenfluss vom Laden eines Profils bis zur UI-Aktualisierung.

Architektur-Diagramm




LoadedArchetypProfile.ich
{
archetyp: 'single',
profileReview: { flatNeeds: {...} },
← 220 Bedürfnisse
gewichtungen: { O, A, D, G },
← Faktor-Gewichte
resonanzFaktoren: { R1, R2, R3, R4 }
← Resonanzwerte
}
SCHRITT 2: MemoryManager holt berechnete Werte
const loadedProfile = LoadedArchetypProfile.ich;
gewichtungen = data.gewichtungen || loadedProfile.gewichtungen;
resonanzFaktoren = data.resonanzfaktoren || loadedProfile.resonanzFaktoren;
SCHRITT 3: UI-Aktualisierung
applyGewichtungen(gewichtungen, 'ich')
→ Speichert in: tiage_faktor_gewichtungen_ich
applyResonanzfaktoren(resonanzFaktoren, 'ich')
→ Speichert in: tiage_resonanz_faktoren_ich
ResonanzCard.setCalculatedValues(resonanzValues, false)
→ Aktualisiert Slider-UI (R1-R4)

Detaillierte Funktionsbeschreibungen

1. ProfileCalculator.loadProfile(person, storageData)

> **Datei:** `profiles/archetypen/index.js`
> **Zeilen:** 286-303
> **Zweck:** Lädt und berechnet ein vollständiges Profil aus Storage-Daten.

```
function loadProfile(person, storageData) {
    // person: 'ich' oder 'partner'
    // storageData: Objekt mit archetyp, geschlecht, dominanz, orientierung

    const calculatedProfile = calculateProfile(storageData);
    window.LoadedArchetypProfile[person] = calculatedProfile;

    return true;
}
```

Eingabe-Parameter:

Parameter	Typ	Beschreibung
`person`	`string`	`'ich'` oder `'partner'`
`storageData`	`Object`	Profildaten aus Storage

storageData Struktur:

```
{
    archetyp: 'single',           // Archetyp-Key
    geschlecht: {                 // Geschlechts-Dimension
        primary: 'mann',
        secondary: 'cis'
    },
    dominanz: {                   // Dominanz-Dimension
        primary: 'switch',
```

```
        secondary: null
    },
    orientierung: {               // Orientierungs-Dimension
        primary: 'hetero',
        secondary: null
    },
    gewichtungen: {...},          // Optional: Gespeicherte Gewichtungen
    resonanzfaktoren: {...}       // Optional: Gespeicherte Resonanzfaktoren
}
```

---

2. calculateFlatNeeds(archetyp, geschlecht, dominanz, orientierung)

> **Datei:** `profiles/archetypen/index.js`
> **Zeilen:** 158-192
> **Zweck:** Berechnet die 220 flatNeeds-Werte aus Basis-Profil + Modifier.

```
function calculateFlatNeeds(archetyp, geschlecht, dominanz, orientierung) {
    // 1. Basis-Bedürfnisse aus BaseArchetypProfile
    const baseProfil = window.BaseArchetypProfile[archetyp];
    const flatNeeds = { ...baseProfil.beduerfnisse };

    // 2. Modifier berechnen und anwenden
    const profileContext = {
        geschlecht: geschlecht,
        dominanz: dominanz?.primary || dominanz,
        orientierung: orientierung?.primary || orientierung
    };

    const deltas = ProfileModifiers.calculateProfileDeltas(profileContext);

    // 3. Deltas anwenden (0-100 begrenzt)
    Object.keys(deltas).forEach(key => {
        flatNeeds[key] = Math.min(100, Math.max(0, flatNeeds[key] + deltas[key]));
    });

    return flatNeeds;
}
```

Berechnungsformel:

```
flatNeed[i] = BaseArchetypProfile[archetyp].beduerfnisse[i]
            + GenderModifier[i]
            + DominanzModifier[i]
            + OrientierungModifier[i]

Begrenzt auf: 0 ≤ flatNeed[i] ≤ 100
```

3. calculateResonanzFaktoren(profileContext)

> **Datei:** `profiles/archetypen/index.js`
> **Zeilen:** 200-231
> **Zweck:** Berechnet die Resonanzfaktoren R1-R4 aus dem Profil-Kontext.

```
function calculateResonanzFaktoren(profileContext) {
    // Prüfe ob TiageSynthesis verfügbar
    if (!TiageSynthesis?.NeedsIntegration?.calculateDimensionalResonance) {
        return getDefaultResonanzFaktoren();
    }

    // Berechne dimensionale Resonanzen
    const resonanz = TiageSynthesis.NeedsIntegration
        .calculateDimensionalResonance(profileContext);

    // Mapping: R1=leben, R2=philosophie, R3=dynamik, R4=identitaet
    return {
        R1: { value: resonanz.leben || 1.0, locked: false },
        R2: { value: resonanz.philosophie || 1.0, locked: false },
        R3: { value: resonanz.dynamik || 1.0, locked: false },
        R4: { value: resonanz.identitaet || 1.0, locked: false }
    };
}
```

Resonanzfaktoren-Mapping:

Faktor	Dimension	Beschreibung	Wertebereich
R1	Leben	Existenz, Zuneigung, Muße	0.5 - 1.5
R2	Philosophie	Freiheit, Teilnahme, Identität	0.5 - 1.5
R3	Dynamik	Dominanz, Sicherheit	0.5 - 1.5
R4	Identität	Verständnis, Erschaffen, Verbundenheit	0.5 - 1.5

4. MemoryManager.applyMeData(data)

> **Datei:** `js/memory-manager.js`
> **Zeilen:** 646-756
> **Zweck:** Wendet geladene Daten auf das ICH-Profil an und aktualisiert die UI.

```
function applyMeData(data) {
    // 1. Profil berechnen und in LoadedArchetypProfile laden
    ProfileCalculator.loadProfile('ich', data);

    // 2. TiageState aktualisieren
    TiageState.set('personDimensions.ich.geschlecht', data.geschlecht);
    TiageState.set('personDimensions.ich.dominanz', data.dominanz);
    TiageState.set('personDimensions.ich.orientierung', data.orientierung);

    // 3. Berechnete Werte aus LoadedArchetypProfile holen
    const loadedProfile = window.LoadedArchetypProfile?.ich;

    // Priorisierung: Storage > Berechnet
    const gewichtungen = data.gewichtungen || loadedProfile?.gewichtungen;
    const resonanzFaktoren = data.resonanzfaktoren || loadedProfile?.resonanzFaktor

    // 4. UI aktualisieren
    applyGewichtungen(gewichtungen, 'ich');
    applyResonanzfaktoren(resonanzFaktoren, 'ich');

    // 5. ResonanzCard UI aktualisieren
    ResonanzCard.setCalculatedValues({
        R1: resonanzFaktoren.R1?.value || 1.0,
        R2: resonanzFaktoren.R2?.value || 1.0,
        R3: resonanzFaktoren.R3?.value || 1.0,
        R4: resonanzFaktoren.R4?.value || 1.0
    }, false);

    return true;
}
```

Datenstrukturen

LoadedArchetypProfile

> **Speicherort:** `window.LoadedArchetypProfile`

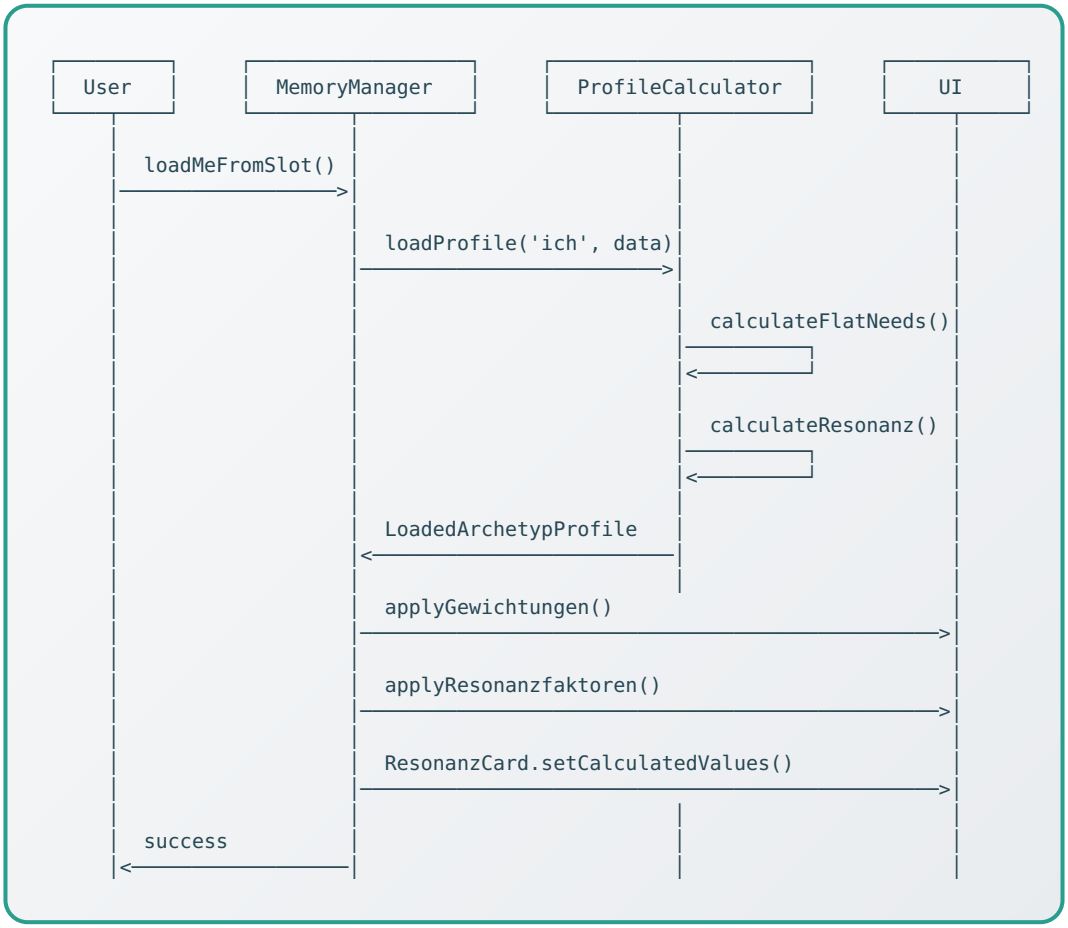
```
{
    ich: {
        archetyp: 'single',
        geschlecht: { primary: 'mann', secondary: 'cis' },
        dominanz: { primary: 'switch', secondary: null },
        orientierung: { primary: 'hetero', secondary: null },
        profileReview: {
            flatNeeds: {
                '#B1': 75,    // 220 Bedürfnisse
                '#B2': 60,
                // ...
            }
        },
        gewichtungen: {
            O: { value: 25, locked: false },  // Orientierung
            A: { value: 25, locked: false },  // Archetyp
            D: { value: 25, locked: false },  // Dominanz
            G: { value: 25, locked: false }   // Geschlecht
        },
        resonanzFaktoren: {
            R1: { value: 1.0, locked: false },
            R2: { value: 1.0, locked: false },
            R3: { value: 1.0, locked: false },
            R4: { value: 1.0, locked: false }
        }
    },
    partner: { ... }  // Gleiche Struktur
}
```

LocalStorage Keys

Key	Beschreibung
`tiage_faktor_gewichtungen_ich`	Gewichtungen für ICH

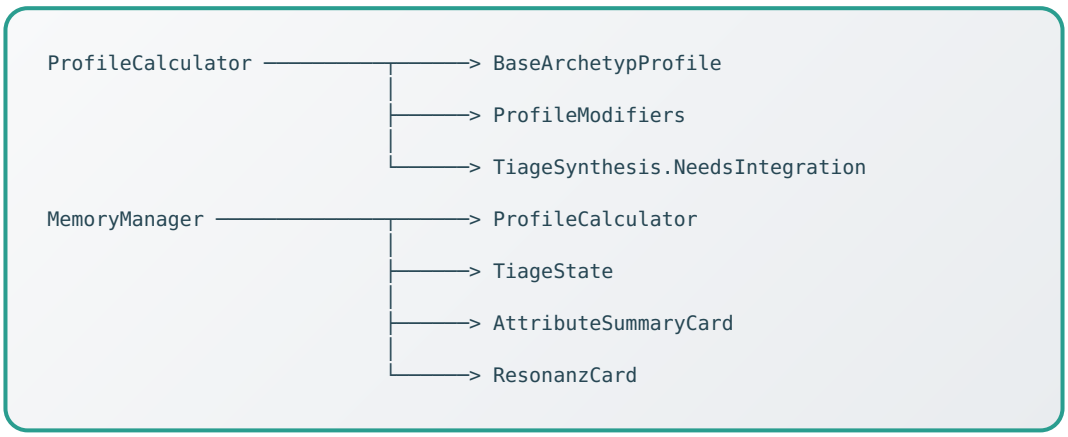
`tiage_faktor_gewichtungen_partner`	Gewichtungen für Partner
`tiage_resonanz_faktoren_ich`	Resonanzfaktoren für ICH
`tiage_resonanz_faktoren_partner`	Resonanzfaktoren für Partner
`tiage_memory_ME001` - `ME004`	Memory-Slots für ICH
`tiage_memory_PART001` - `PART004`	Memory-Slots für Partner

Sequenzdiagramm



Abhängigkeiten

Erforderliche Module



Ladeketten

```
1. BaseArchetypProfile (Basis-Definitionen)
   ↓
2. ProfileModifiers (Modifier-Definitionen)
   ↓
3. TiageSynthesis (Berechnungslogik)
   ↓
4. ProfileCalculator (Haupt-Orchestrator)
   ↓
5. MemoryManager (Storage + UI-Bridge)
   ↓
6. ResonanzCard / AttributeSummaryCard (UI-Komponenten)
```

Fehlerbehandlung

Fallback-Werte

Komponente	Fallback
flatNeeds	Leeres Objekt `{}`

gewichtungen	`{ O: 25, A: 25, D: 25, G: 25 }`
resonanzFaktoren	`{ R1: 1.0, R2: 1.0, R3: 1.0, R4: 1.0 }`

Validierung

- Archetyp-Key muss in `BaseArchetypProfile` existieren
- flatNeeds-Werte werden auf 0-100 begrenzt
- Resonanzfaktoren werden auf 0.5-1.5 begrenzt

Verwandte Dokumentation

- README.md - Hauptdokumentation
- theory/resonance.md - Resonanz-Theorie
- theory/factors.md - Die 4 Qualitätsfaktoren
- NAMING_CONVENTION.md - ID-Referenzsystem
